

# HoGent

BEDRIJF  
EN  
ORGANISATIE

## Databases II – SQL

Join - Subqueries, Insert-Update-Delete, Views  
Common Table Expressions

# **JOIN**

**working with  
more than one table**

# SQL - DML

- Consult one table
- **Consult > 1 table**
  - JOIN
    - Inner join
    - Outer join
    - Cross join
  - UNION
  - Subquery's
    - Simple nested query's
    - Gecorrelated subquery's
    - Operator EXISTS
  - Set Operators

# JOIN

- **JOIN**

- Select columns from several tables
  - JOIN keyword : specifies which tables have to be joined and how
    - Inner join
    - Outer join
    - Cross join
  - ON keyword : specifies the JOIN condition
- Produces 1 result set, joining the rows of both tables
- Basic form (ANSI JOIN (SQL-92) <-> Old style join)

```
SELECT expression  
FROM table1 JOIN table2 ON condition  
      [JOIN table2 ON condition...]
```

```
SELECT expression  
FROM table1, table2 [, table3...]  
WHERE condition
```

# INNER JOIN

- Joins rows from one table with rows from other table based on common criteria in the corresponding tables.
- The relation between the fields in the corresponding tables is expressed through:
  - = (equi-join)
  - <
  - >
  - <>
  - >=
  - <=

# INNER JOIN

- Example of equi-join
  - Give the team number and name of the captain of each team
    - ANSI JOIN (SQL-92)

```
SELECT TEAMNO, NAME  
FROM TEAMS JOIN PLAYERS  
ON TEAMS.PLAYERNO = PLAYERS.PLAYERNO
```

- OR “old style join”

```
SELECT TEAMNO, NAME  
FROM TEAMS , PLAYERS  
WHERE TEAMS.PLAYERNO = PLAYERS.PLAYERNO
```

# Aliases

- USE tables aliases (via 'AS' or blank)

- SQL-92

```
SELECT T.TEAMNO, NAME  
FROM TEAMS T JOIN PLAYERS P  
ON T.PLAYERNO = P.PLAYERNO
```

- “old style join”

```
SELECT T.TEAMNO, NAME  
FROM TEAMS T , PLAYERS P  
WHERE T.PLAYERNO = P.PLAYERNO
```

## Remarks:

If the same column name is used in several tables in a query, then each column name has to be preceded by the table name or its alias.

Inner joins only return rows that meet the ON condition.

If you omit (forget) the where clause in the old style join all combinations are returned (= cartesian product) = CROSS JOIN (see further)

# INNER JOIN of > 1 tables

- JOIN of more than 2 tables

- Example: give for all matches the number, number and name of the player, team number and name of the division
- SQL-92 :

```
SELECT M.MATCHNO,M.PLAYERNO,  
M.TEAMNO,P.NAME,T.DIVISION  
FROM MATCHES M JOIN PLAYERS P ON  
M.PLAYERNO=P.PLAYERNO  
JOIN TEAMS T ON M.TEAMNO=T.TEAMNO
```

- Old style join

```
SELECT M.MATCHNO,M.PLAYERNO,M.TEAMNO,P.NAME,T.DIVISION  
FROM MATCHES M, PLAYERS P,TEAMS T  
WHERE M.PLAYERNO=P.PLAYERNO  
AND M.TEAMNO=T.TEAMNO
```



# INNER JOIN of a table with itself

- Example: Show all employees and the name of their manager (db xtreme)

```
select e1.employeeID,  
e1.lastname & ' ' & e1.firstname as werknemer,  
e2.lastname & ' ' & e2.firstname as baas  
from employee e1 inner join employee e2  
on e1.reportsto=e2.employeeid;
```

# OUTER JOIN

- Returns all records from 1 table, even if there is no corresponding record in the other table
- 3 types of an outer join
  - LEFT OUTER JOIN
    - Returns all rows of the first table in the FROM clause (SQL-92)
  - RIGHT OUTER JOIN
    - Returns all rows of the second table in the FROM clause (SQL-92)
  - FULL OUTER JOIN
    - Returns all rows of the first and the second table in the FROM clause (SQL-92) even if there is no corresponding record in the other table

# LEFT OUTER JOIN

- Example: give for all players their player number and name and the list of their penalties (db TENNIS)

```
SELECT p.playerno, name, amount  
FROM players p LEFT OUTER JOIN penalties pe  
ON p.playerno=pe.playerno  
ORDER BY p.playerno
```

# RIGHT OUTER JOIN

- Example: give for all players the player number, the name and the number of matches played by that player.

```
select p.playerno,p.name,count(m.matchno)
from matches m right join players p
on m.playerno=p.playerno
group by p.playerno,p.name
order by p.playerno;
```

spelersnr	naam	count(w.wedstrijdnr)
2	Elfring	1
6	Pementier	3
7	Wijers	0
8	Niewenburg	2
27	Cools	1
28	Cools	0
39	Bischoff	0

- Pay attention: COUNT(\*) shows all lines

```
select p.playerno,p.name,count(*)
from matches m right join players p
on m.playerno=p.playerno
group by p.playerno,p.name
order by p.playerno;
```

spelersnr	naam	count(*)
2	Elfring	1
6	Pementier	3
7	Wijers	1
8	Niewenburg	2
27	Cools	1
28	Cools	1
39	Bischoff	1

# CROSS JOIN

- In a cross join the number of rows in the result set equals the number of rows in the first table multiplied by the number of rows in the second table
- Application: generate all combinations
- Example. Make a competition schedule in which each player plays once against each other player

– SQL-92

```
select p1.name+' '+p1.initials PLAYER1, '-' AGAINST, p2.name+' '+p2.initials PLAYER2
from players p1 cross join players p2
where p1.playerno < p2.playerno
order by p1.playerno;
```

– old style join

```
select p1.name+' '+p1.initials PLAYER1, '-' AGAINST, p2.name+' '+p2.initials PLAYER2
from players p1, players p2
where p1.playerno < p2.playerno
order by p1.playerno;
```

# **SET OPERATORS: UNION**

# UNION

- With a **UNION** you combine the result of two or more queries
  - Basic form

```
SELECT ... FROM ... WHERE ...  
UNION  
SELECT ... FROM ... WHERE ...  
ORDER BY ...
```

- Rules
  - Both SELECTs have to contain an equal number of columns
  - Corresponding columns from both SELECTs should have compatible data types
  - The columns names or aliases from the first SELECT are shown
  - The result set does not contain duplicates. To keep duplicates use UNION ALL
  - At the end an ORDER BY can be added. Column names or expressions can't be used in the ORDER BY if they differ between the two SELECTs. In this case use column numbers for sorting.

- Example: give an overview of all employees (lastname and firstname, city and postal code) and all customers (name, city and postal code)

```
SELECT lastname + ' ' + firstname as name, city, postalcode
FROM Employee
UNION
SELECT customername, city, postalcode
FROM Customer
```

name	city	postalcode
Davolio Nancy	Port Moody	V3D 4F6
Fuller Andrew	Coquitlam	V3H4J7
Leverling Janet	Vancouver	V6M 8S9
Peacock Margaret	Richmond	V5S 6H7
Buchanan Steven	London	SW1 8JR
Suyama Michael	London	EC2 7JR
King Robert	London	RG1 9SP
Callahan Laura	New Westminster	V7J 5G5
Dodsworth Anne	Nottingham	WG2 7LT
Hellstern Albert	Burnaby	V2C 8H3
Smith Tim	North Vancouver	V3K 2G9



# INTERSECT

- Which records are in the intersection?

```
select city, country from customer  
intersect  
select city, country from supplier
```

# EXCEPT

- The **EXCEPT** operator subtracts a result set from another result set.
  - Example: which products have never been ordered?

```
SELECT productid FROM product  
EXCEPT  
SELECT productid FROM ordersdetail;
```

# Some exercises

## **Database Xtreme:**

1. Which suppliers (Id and name) deliver class Bicycle products?
2. Give for each supplier the number of orders that contain products of that supplier. Show supplierID, supplier name and the number of orders. Order by supplier name.
3. What's for each type the lowest productprice of products for Youth? Show producttype name and lowest. Hint: products for youth contain "Y" in the M\_F field
4. Give for each purchased productId: productname, the least and the most ordered. Order by productname.
5. Give a summary for each employee with orderID, employeeID and employeename. Make sure that the list also contains employees who don't have orders yet.

## **Database tennis:**

6. Give name, initials and total penalty amount of each player that already got a total of more than 150 euro of penalties

# Some exercises

## Database tennis:

Make a report that shows all actual board members. Show the following information, ordered by name:

- player number and name
- complete address data as a single column in the output. Use commas and blanks where appropriate
- their age in the year they became a member of the club
- their exact age, expressed in years, on the day their actual board period started.

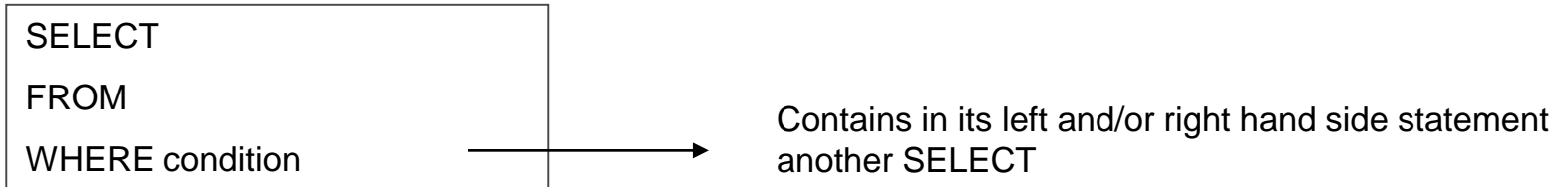
Give each column an appropriate name and export it to Excel.

NUM... ▾	NAAM	ADRES	Leeftijd als lid	Leeftijd als bestuurslid
2	Everett	Stoney Road 43, 3575NH Stratford	27	22
6	Parmenter	Haseltine Lane 80, 1234KK Stratford	13	23
8	Newcastle	Station Road 4, 6584WO Inglewood	18	22
95	Miller	High Street 33A, 5746OP Douglas	9	22
112	Bailey	Vixen Road 8, 6392LK Plymouth	21	22

# **SUBQUERIES**

# SUBQUERIES basic form

- Nested subqueries
  - Basic form



- Outer level query = the first SELECT. This is the main question
- Inner level query = the SELECT in the WHERE clause (or HAVING clause). This is the sub query:
  - Always executed first
  - Always between ().
  - Subqueries can be nested at > 1 level.
- A subquery can
  - return one value
  - return a list of values

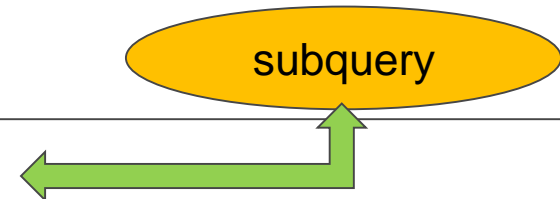
# SUBQUERY that returns a single value

- The result of the query can be used anywhere you can use an expression.
  - With all relational operators: =, >, <, <=, >=, <>
  - Example:
    - What is the highest salary? (db xTreme)

```
select max(salary)
from employee
```

- Who has the highest salary?

```
select lastname,firstname,salary
from employee
where salary =
(select max(salary) from employee)
```



First the table employee is searched to determine the highest salary (= subquery). Then the table is searched a second time (= main query) to evaluate each employee's salary against the determined maximum.

# SUBQUERY that returns a single value

- Other examples
  - Determine the salary of the employees that earn more than average

```
select lastname,firstname,salary
from employee
where salary >(select avg(salary) from employee)
```

- Who is the youngest employee from Canada?

```
select lastname,firstname
from employee
where country='Canada'
and birthdate =(select max(birthdate) from employee where country='Canada');
```



# SUBQUERY that returns a single column

- the resulting column can be used as a list
  - Operators IN, NOT IN, ANY, ALL
  - IN operator (=ANY operator)
    - » DB Tennis: give all players that played matches (can also be accomplished with join)

```
SELECT playerno  
from players  
where playerno in (select playerno from matches)
```

- » Give name of the players who live in the same town as R. Permenter

```
select name  
from players  
where town= (select town from players where name='Parmenter' and initials='R');
```

- » Why is the query below not working?

```
select name from players  
where town=(select town from players where name ='Permenter');
```

# SUBQUERY that returns a single column

- NOT IN / <>ALL operator
  - » Give all players that did not play any matches

```
select playerno  
from players  
where playerno not in (select playerno from matches)
```

- » This can't be solved with INNER JOIN, only with OUTER JOIN and EXISTS (see below)

# ANY and ALL keywords

- These keywords are used in combination with the relational operators and subqueries that return a column of values
  - **ALL** returns TRUE if all values returned in the subquery satisfy the condition
  - **ANY** returns TRUE if at least one value returned in the subquery satisfies the condition
  - Example: give the highest playerno and the corresponding leagueno.

```
SELECT PLAYERNO, LEAGUENO
FROM PLAYERS
WHERE PLAYERNO >= ALL (SELECT PLAYERNO FROM PLAYERS WHERE LEAGUENO IS NOT NULL);
```

- Example: Give the playernos with at least one penalty that is larger than a penalty paid by player 27; player 27 himself should not appear in the result.

```
SELECT DISTINCT PLAYERNO
FROM PENALTIES
WHERE PLAYERNO <> 27
AND AMOUNT > ANY (SELECT AMOUNT FROM PENALTIES WHERE PLAYERNO=27);
```

# ANY and ALL keywords

```
SELECT PLAYERNO, LEAGUENO  
FROM PLAYERS  
WHERE PLAYERNO >= ALL (SELECT PLAYERNO FROM PLAYERS WHERE LEAGUENO IS NOT NULL);
```

Returns the same result as:

```
SELECT PLAYERNO, LEAGUENO  
FROM PLAYERS  
WHERE PLAYERNO = (SELECT MAX(PAYERNO) FROM PLAYERS WHERE LEAGUENO IS NOT NULL);
```

```
SELECT DISTINCT PLAYERNO  
FROM PENALTIES  
WHERE PLAYERNO <> 27  
AND AMOUNT > (SELECT MIN(AMOUNT) FROM PENALTIES WHERE PLAYERNO=27);
```

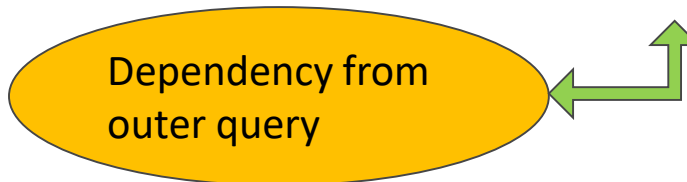
Returns the same result as:

```
SELECT DISTINCT PLAYERNO  
FROM PENALTIES  
WHERE PLAYERNO <> 27  
AND AMOUNT > ANY (SELECT AMOUNT FROM PENALTIES WHERE PLAYERNO=27);
```

# Correlated subqueries

- In a correlated subquery **the inner query depends on information from the outer query.**
  - the contains a search condition that refers to the main query, which make the subquery depends on the main query
- The subquery is executed for each row in the main query.
  - $O(n^2)$
  - The order of execution is from top to bottom, not from bottom to top as in a simple subquery, which is  $O(n)$ .
- For performance reasons use joins or simple subquery is possible
- Principle

```
SELECT ...  
FROM table a  
WHERE expression operator (SELECT ...  
                           FROM table  
                           WHERE expression operator a.columnname)
```



# Correlated subqueries

- Example: give employees with a salary larger than the average salary (db Xtreme)

```
SELECT lastname, firstname, salary
FROM employee
WHERE salary >
(
    SELECT AVG(salary)
    FROM employee
);
```

- Give the employees whose salary is larger than the average of the salary of the employees who report to the same boss.

```
SELECT lastname, firstname, salary
FROM employee AS e
WHERE salary >
(
    SELECT AVG(salary)
    FROM employee
    WHERE reportsto = e.reportsto
);
```

**Remark: in the inner query you can use fields from the tables in the outer query but NOT vice versa.**

0. Row 1 in the outer query

1. Outer query passes column values for that row to inner query

2. Inner query use those values to evaluate inner query.

3. Inner query returns value to outer query, which decides if row in outer query will be kept.

4. This process repeats for each row in outer query.



Back to step 1.

# Subqueries and the EXISTS operator

- The operator EXISTS tests the existence of a result set.
- There is also NOT EXISTS
  - Example: give the players that did not play any matches yet.

```
SELECT *  
FROM players AS p  
WHERE NOT EXISTS  
(  
    SELECT * FROM matches  
    WHERE playerno = p.playerno  
);
```

- Give the players that did play matches

```
SELECT *  
FROM players AS p  
WHERE EXISTS  
(  
    SELECT * FROM matches  
    WHERE playerno = p.playerno  
);
```

# 3 ways to accomplish the same result

Who did not play any matches?

-- OUTER JOIN

```
SELECT p.playerno
FROM players AS p
      LEFT JOIN
      matches AS m
ON p.playerno = m.playerno
WHERE m.playerno IS NULL;
```

-- SIMPLE SUBQUERY

```
SELECT playerno FROM players
WHERE playerno NOT IN
(
  SELECT playerno FROM matches
);
```

-- CORRELATED SUBQUERY

```
SELECT playerno
FROM players AS p
WHERE NOT EXISTS
(
  SELECT * FROM matches
  WHERE playerno = p.playerno
);
```



# Subqueries in the FROM-clause

- Since the result of a query is a table it can be used in the FROM-clause.
- In MS-SQL Server the table in the subquery must have a name. You can optionally also rename the columns
  - Example: give per region (USA+Canada=North America, rest=Rest of World) the total sales.
- A subquery in the FROM clause is called a *derived table*.

```
-- Solution 1
select
case c.country
when 'USA' then 'Northern America'
when 'Canada' then 'Northern America'
else 'Rest of world'
end as regionclass, sum(orderamount)
from customer c join orders o
on c.CustomerID=o.CustomerID
group by
case c.country
when 'USA' then 'Northern America'
when 'Canada' then 'Northern America'
else 'Rest of world'
end ;
-- drawback: copy-paste of case
```

```
-- Solution 2
-- avoid copy-paste via subquery in FROM
select region, sum(total) from
(
select
case c.country
when 'USA' then 'Northern America'
when 'Canada' then 'Northern America'
else 'Rest of world'
end as region, orderamount
from customer c join orders o
on c.CustomerID=o.CustomerID
)
as totals(regionclass,total)
group by region;
```

# Subqueries in the SELECT-clause

- In a SELECT clause scalar (simple or correlated) subqueries can be used
  - E.g. give for each employee how much they earn more (or less) than the average salary of all employees with the same supervisor.

```
SELECT lastname, firstname, salary,  
salary -  
(  
    SELECT AVG(salary)  
    FROM employee  
    WHERE supervisorid = e.supervisorid  
)  
FROM employee e;
```

# Subqueries in the SELECT- and FROM-clause

- (db xtreme): give per productclass the price of the cheapest product and a product that has that price.

```
SELECT class, unitprice,  
(  
    SELECT TOP 1 productid  
    FROM product  
    WHERE productclassid = class AND  
           price = unitprice  
)  
FROM  
(  
    SELECT productclassid, MIN(price)  
    FROM product AS p  
    GROUP BY productclassid  
) AS pcmin(class, unitprice);
```

# Application: running totals

Running total of orderamount per year:

```
SELECT orderid,orderdate,orderamount,  
(select sum(orderamount)  
from orders where year(orderdate) =  
year(o.orderdate) and orderid <= o.orderid)  
FROM orders o  
order by orderid;
```

# Application: monthly gross margin

```
SELECT isnull(ord.month, pur.month), isnull(ord.amount, 0) - isnull(pur.amount, 0) AS margin
FROM(
  (
    SELECT format(orderdate, 'yyyy-MM'), SUM(orderamount)
    FROM orders
    GROUP BY format(orderdate, 'yyyy-MM')
  ) AS ord(month, amount)
FULL JOIN
  (
    SELECT format(orderdate, 'yyyy-MM'), SUM(p.price * pu.UnitsOnOrder)
    FROM purchases AS pu
    JOIN
      product AS p
    ON pu.PRODUCTID = p.PRODUCTID
    GROUP BY format(orderdate, 'yyyy-MM')
  ) AS pur(month, amount)
ON ord.month = pur.month
ORDER BY 1;
```

# Some exercises

## Database Xtreme:

1. Give the id and name of the products that have not been purchased yet.
2. Select the names of the suppliers who supply products that have not been ordered yet.
3. Select the products (all data) with a price that is higher than the average price of the "Bicycle" products. Select in descending order of price.
4. Show a list of the orderID's of the orders for which the order amount differs from the amount calculated through the ordersdetail.
5. Which employee has processed most orders?
6. Give per employee and per order date the total order amount. Also add the name of the employee and the running total per employee when ordering by orderdate:

EMPLOYEEID	LASTNAME	FIRSTNAME	ORDERDATE	SOM	RUNNING
1	Davolio	Nancy	19/02/2000	848	848
1	Davolio	Nancy	26/02/2000	69	916
1	Davolio	Nancy	27/02/2000	5308	6224
1	Davolio	Nancy	2/12/2000	42	6266
1	Davolio	Nancy	3/12/2000	25131	31398
1	Davolio	Nancy	4/12/2000	29	31427
1	Davolio	Nancy	7/12/2000	8710	41137

## Database tennis:

7. Give the name and number of the players that already got more penalties than they played matches.

**INSERT – UPDATE –  
DELETE**

**INSERT**

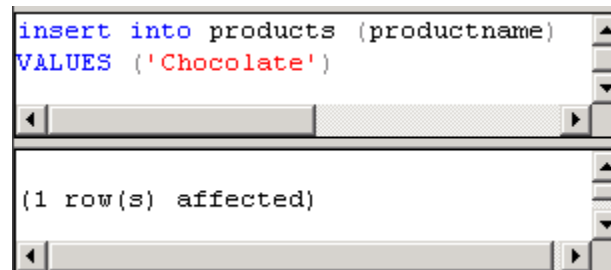


# SQL - DML basic tasks

- SELECT  
consulting data
- INSERT  
adding data
- UPDATE  
changing data
- DELETE  
removing data

# Adding data - INSERT

- The **INSERT** statement adds data in a table
  - Add one row through via specification
  - Add selected row(s) from other tables



```
insert into products (productname)
VALUES ('Chocolate')
```

(1 row(s) affected)

The image shows a screenshot of a SQL query execution window. The top pane contains the SQL statement: `insert into products (productname)` followed by `VALUES ('Chocolate')` on the next line. The bottom pane displays the result: `(1 row(s) affected)`. Both panes have horizontal scrollbars.

# INSERT of 1 row

- Example: Add product “Chocolate” with category 1
  - method 1: specify only the (not NULL) values for specific columns

```
INSERT INTO product (productname, categoryid)  
VALUES ( 'Chocolate',1)
```

- method 2: specify all column values

```
INSERT INTO product  
VALUES ('Chocolate', NULL, 1, NULL, NULL, NULL, NULL, NULL, 1)
```

# INSERT of 1 row

	Column Name	Data Type	Allow Nulls
▶	ProductID	int	<input type="checkbox"/>
	ProductName	nvarchar(40)	<input type="checkbox"/>
	SupplierID	int	<input checked="" type="checkbox"/>
	CategoryID	int	<input checked="" type="checkbox"/>
	QuantityPerUnit	nvarchar(20)	<input checked="" type="checkbox"/>
	UnitPrice	money	<input checked="" type="checkbox"/>
	UnitsInStock	smallint	<input checked="" type="checkbox"/>
	UnitsOnOrder	smallint	<input checked="" type="checkbox"/>
	ReorderLevel	smallint	<input checked="" type="checkbox"/>
	Discontinued	bit	<input type="checkbox"/>
			<input type="checkbox"/>

The number of specified columns corresponds to the number of values

The specified values and corresponding columns have compatible data types.

If no column names are specified the values are assigned in the column order as specified by the CREATE TABLE statement

Unmentioned columns get the value NULL or the DEFAULT value if any.

NULL can also be specified as a value

# INSERT of row(s) selected from other tables

- Examples: add all employees to the customer table

```
INSERT INTO customers
```

```
SELECT substring(firstname,1,3) + substring(lastname,1,3), lastname,  
firstname, title, address, city, region, postalcode, country, homephone, null  
FROM Employees
```

Mandatory fields have to be specified, unless they have a DEFAULT value.

Constraints are validated

Unmentioned columns get the value NULL or the DEFAULT value if any.

**UPDATE**

# Changing data - UPDATE

- Changing all rows in a table
  - example: increase the price of all products with 10%
- Changing 1 row or a group of rows
  - example: increase the price of the product “Chocolate” with 10%

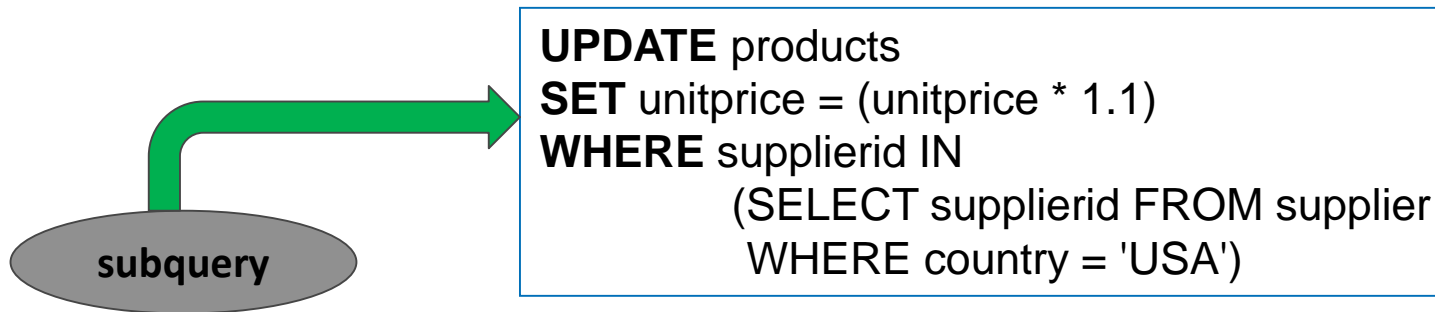
```
UPDATE Products  
SET unitprice = (unitprice * 1.1)
```

```
UPDATE products  
SET unitprice = (unitprice * 1.1)  
WHERE productname = 'Chocolate'
```

```
UPDATE products  
SET unitprice = (unitprice * 1.1), unitsinstock = 0  
WHERE productname = 'Chocolate'
```

# Changing data - UPDATE

- Change rows based on data in another table
  - example: increase the unit price of products delivered by suppliers from the USA with 10%.





**DELETE**

# Removing data - DELETE

- Deleting rows

- example: delete product 'Chocolate'

```
DELETE FROM product  
WHERE productname = 'Chocolate'
```

- Delete all rows in a table

- via DELETE the identity values continues

```
DELETE FROM product
```

- via TRUNCATE restarts from 1 (more performant)

```
TRUNCATE TABLE product
```

# DELETE - based on data in another table

- example: delete the orderdetails of the orders dated 14th of April 1998

```
DELETE FROM [order details]
WHERE orderid IN
    (SELECT orderid
     FROM orders
     WHERE orderdate = '4/14/98')
```

# VIEWS

# views - introduction

- definition
  - A view is a saved SELECT statement
  - A view can be seen as a virtual table composed of other tables & views
    - No data is stored in the view itself, at each referral the underlying SELECT is re-executed;
- advantages
  - hide complexity of the database
    - Hide complex database design
    - make large and complex queries accessible and reusable
    - Can be used as a partial solution for complex problems
  - Used for securing data access: revoke access to tables and grant access to customised views.
  - organise data for export to other applications

# Definition of a view

```
CREATE VIEW view_name [(column_list)]  
AS select_statement  
[with check option]
```

*syntax of CREATE VIEW*

- # nr of columns in (column\_list) = # columns in select
  - If no column names are specified, they are taken from the select
  - Column names are mandatory if the select statement contains calculations or joins in which some column names appear more than once
- The select statement may not contain an order by
- with check option: in case of mutation through the view (insert, update, delete) it is checked if the new data also conforms to the view conditions

# views - CRUD operations

```
CREATE VIEW V_ProductsCustomer(productcode, company, quantity)
AS SELECT productid, companyname, sum(quantity)
   FROM customers join orders ON orders.customerid =
        customers.customerid join [order details] ON orders.orderid
        = [order details].orderid
   GROUP BY productid,companyname
```

*example: creation of a view*

```
SELECT * FROM V_ProductsCustomer
```

*example: use of a view*

```
ALTER VIEW V_ProductsCustomer
AS SELECT productid, companyname
   FROM customers join orders ON orders.customerid =
        customers.customerid join [order details] ON orders.orderid
        = [order details].orderid
   GROUP BY productid,companyname
```

*example: changing a view*

```
DROP VIEW V_ProductsCustomer
```

*example: deleting a view*

# Example: views as partial solution for complex problems

Gross margin problem with views iso subqueries:

```
create view sales(month,total) as
select format(orderdate, 'yyyy-MM'), sum(orderamount)
from orders
group by format(orderdate, 'yyyy-MM');
```

```
create view pur(month,total) as
select format(orderdate, 'yyyy-MM'), sum(p.price*pu.UnitsOnOrder)
from purchases pu join product p on pu.PRODUCTID = p.PRODUCTID
group by format(orderdate, 'yyyy-MM');
```

```
select isnull(s.month,p.month) MONTH, isnull(s.total,0)-
isnull(p.total,0) MARGIN
from sales s full join pur p
on s.month=p.month
order by 1;
```



# update of views

- an **updatable view**
  - Has no distinct or top clause in the select statement
  - Has no statistical functions in the select statement
  - Has no calculated value in the select statement
  - Has no group by in the select statement
  - Does not use a union
- All other views are read-only views

# working with updatable views

- **UPDATE**

- You can only update one table at once
- without check option
  - After the update a row can disappear from the view
- with check option
  - An error is generated if after the update the row would no longer be part of the view

# werken met updatable views

- **INSERT**

- You can only insert in one table
- All mandatory columns have to appear in the view and the insert
  - identity columns with a NULL or DEFAULT constraint can be omitted

- **DELETE**

- the delete can only be used with a VIEW based on exactly one table.


# Views with/without check option: example

```
CREATE VIEW productsOfCategory6  
AS SELECT *  
FROM product  
WHERE categoryID = 6
```

*example: create view without "with check option"*

```
INSERT INTO productsOfCategory6 (productname, categoryid)  
VALUES ('Chocolate', 1)
```

*example: insert product from category 1*



*Although 'Chocolate' does not belong to category 6,  
it can be added through the view*

```
CREATE VIEW productsOfCategory6Bis  
AS SELECT *  
FROM product  
WHERE categoryID = 6  
WITH CHECK OPTION
```

*example: insert statement above generates error message*

# Views in SQL Server Management Studio

- views can also be made with the graphical user interface
  - example: a view with the customers having more than 10 orders

3700-06\SQLEXPR...dbo.TrouweKlant\* 3700-06\SQLEXPRES...d - Alle tabellen

Customers

- ☐ \* (All Columns)
- ☐ CustomerID
- ☐ CompanyName
- ☒ ContactName
- ☐ ContactTitle

Orders

- ☐ \* (All Columns)
- ☒ OrderID
- ☐ CustomerID
- ☐ EmployeeID
- ☐ OrderDate

Column	Alias	Table	Output	Sort Type	Sort Order	Group By	Filter
ContactName	[Trouw...	Customers	<input checked="" type="checkbox"/>			Group By	
CustomerID		Orders	<input type="checkbox"/>			Group By	
*			<input type="checkbox"/>			Count	> 10

```
SELECT dbo.Customers.ContactName AS [Trouwe Klant]
FROM dbo.Customers INNER JOIN
      dbo.Orders ON dbo.Customers.CustomerID = dbo.Orders.CustomerID
GROUP BY dbo.Orders.CustomerID, dbo.Customers.ContactName
HAVING (COUNT(*) > 10)
```

Trouwe Klant
Thomas Hardy
Christina Ber...
Frédérique Ci...
Laurence Le...

1 of 28 | Cell is Read Only.

# Views in SQL Server Management Studio

- The easiest way to change the SQL code of an existing view is by right clicking on the name of the view and:
  - Script View as...
  - ALTER to...
  - New Query Editor Window

# **COMMON TABLE EXPRESSIONS**

# Common Table Expressions: the WITH component

the WITH-component has two application areas:

1. Simplify SQL-instructions, ex. simplified alternative for subqueries or avoid repetition of SQL constructs
2. Traverse recursively hierarchical and network structures

Ex.: give the average number of penalties of all players? This can be solved using a subquery (SELECT AVG(COUNT)) is not possible):

```
SELECT AVG(number * 1.0) -- *1.0 to force floating point
FROM
(
    SELECT COUNT(pe.playerno)
    FROM players AS pl
        LEFT JOIN
            penalties AS pe
        ON pl.PLAYERNO = pe.PLAYERNO
    GROUP BY pl.PLAYERNO
) AS fines(number);
```



# Common Table Expressions: the WITH component

Using the WITH-component you can give the subquery its own name (with column names) and reuse it in the rest of the query (possibly several times):

```
WITH fines(number)
AS (SELECT COUNT(pe.playerno)
     FROM players AS pl
     LEFT JOIN
     penalties AS pe
     ON pl.PLAYERNO = pe.PLAYERNO
     GROUP BY pl.PLAYERNO)
SELECT AVG(number * 1.0)
FROM fines;
```

An expression like this is called:

**common table expression**, shortened as **CTE**

# CTE's vs. Views

- Similarities
  - WITH ~ CREATE VIEW
  - Both are virtual tables: the content is derived from other tables
- Differences
  - A CTE only exists during the SELECT-statement
  - A CTE is not visible for other users and applications

# CTE's vs. Subqueries

- Similarities
  - Both are virtual tables: the content is derived from other tables
- Differences
  - A CTE can be reused in the same query.
  - A subquery is defined in the clause where it is used (SELECT/FROM/WHERE/...)
  - A CTE is defined on top of the query
  - A simple subquery can always be replaced by a CTE

# CTEs to avoid repetition of subqueries

*Give the payment numbers and penalty amount that are not equal to the highest and lowest penalty ever paid by player 44. Also show this highest and lowest amount in the result.*

Without CTE:

```
SELECT      paymentno, amount,
(SELECT MIN(amount) FROM penalties WHERE playerno = 44),
(SELECT MAX(amount) FROM penalties WHERE playerno = 44)
FROM penalties
WHERE amount <>
(SELECT MIN(amount) FROM penalties WHERE playerno = 44)
AND amount <>
(SELECT MAX(amount) FROM penalties WHERE playerno = 44);
```

# CTEs to avoid repetition of subqueries

*Give the payment numbers and penalty amount that are not equal to the highest and lowest penalty ever paid by player 44. Also show this highest and lowest amount in the result.*

With CTE:

```
with min_max(min_amount,max_amount) as
(select min(amount),max(amount) from penalties where
playerno=44)
select p.paymentno,p.amount,
mm.min_amount,mm.max_amount
from penalties p cross join min_max mm
where p.amount <> mm.max_amount and p.amount <>
mm.min_amount;
```

# CTEs to avoid repetition of subqueries

*Generate the numbers 0 to 999*

```
with numbers(number) as
(select 0 as number union select 1 union
select 2 union select 3 union select 4 union
select 5 union select 6 union select 7 union
select 8 union select 9)
select (number1.number * 100) +
(number2.number * 10) + number3.number as number
from numbers as number1 cross join numbers as
number2 cross join numbers as number3
order by number;
```

# CTEs to simplify queries

- (db xtreme): give per productclass the price of the cheapest product and all products with that price.
- Using a subquery:

```
SELECT class, unitprice,  
(  
    SELECT TOP 1 productid  
    FROM product  
    WHERE productclassid = class AND  
           price = unitprice  
)  
FROM  
(  
    SELECT productclassid, MIN(price)  
    FROM product AS p  
    GROUP BY productclassid  
) AS pcmin(class, unitprice);
```

Disadvantage: top 1 is necessary in case several products have that price. As a consequence, only one product per class can be shown .

# CTEs to simplify queries

- (db xtreme): give per productclass the price of the cheapest product and all products with that price.

Solution: with CTE:

```
WITH pcmin(class, unitprice)
AS (SELECT productclassid, MIN(price)
     FROM product AS p
     GROUP BY productclassid)
SELECT class, unitprice, productid
FROM product AS p
JOIN
pcmin AS pc
ON p.ProductClassID = pc.class
WHERE p.price = pc.unitprice;
```

Now we get all products with that price.



# CTE's wit > 1 WITH-component

*What is the total number of rows in both the penalties and the matches table (db tennis)?*

```
with nr_penalties(nr) as
    (select count(*) from penalties),
    nr_matches(nr) as
    (select count(*) from matches)
select (
    (select nr from nr_penalties) +
    (select nr from nr_matches)
);
```

# Recursive SELECTs

- Recursivity means:  
We continue to execute a table expression until a condition is reached.
- This allows you to solve problems like:
  - Who are the friends of my friends etc. (in a social network)
  - What is the hierarchy of an organisation ?
  - Find the parts and subparts of a product (Bill of materials).

# Recursive SELECTs

*Give the integers from 1 to 5*

```
with numbers(number) as
(select 1 union all
 select number + 1 from numbers
 where number < 5)
select * from numbers;
```

Characteristics of recursive use of WITH:

- The with component consists of (at least) 2 expressions, combined with union all
- A temporary table is consulted in the second expression
- At least one of the expressions may not refer to the temporary table.

# Recursive SELECTs: how does it work?

1. SQL searches the table expressions that don't contain recursivity and executes them one by one.

`select 1` → 

	number
1	1

2. Execute all recursive expressions. The numbers table, that got a value of 1 in step 1, is used.

`select number + 1 from numbers`  
`where number < 5`

→ 

	number
1	2

This row is added to the numbers table.

# Recursive SELECTs: how does it work?

3. Now the recursion starts: the 2<sup>nd</sup> expression is re-executed, giving as results:

	number
1	3

Remark: not all rows added in the previous steps are processed, but only those rows (1 row in this example), that were added in the previous step (step 2).

4. Since step 3 also gave a result, the recursive expression is executed again, producing an intermediate result:

	number
1	4

# Recursive SELECTs: how does it work?

5. And this happens again:

	number
1	5

6. If the expressions is now processed again, it does not return a result, since in the previous step now rows were added that correspond to the condition `number < 5`. Here SQL stops the processing of the table expression and the final result is known.

Summary: the 1<sup>st</sup> (non-recursive) expression is executed once and the 2<sup>nd</sup> expression is executed until it does not return any more results.

# Recursive SELECTs : max numbe of recursions = 100

*Give the numbers from 1 to 999 (cf. CTE without recursion)*

```
with numbers(number) as
(select 1 union all
 select number + 1 from numbers
 where number < 999)
select * from numbers;
```

The maximum recursion 100 has been exhausted before statement completion.

# Recursive SELECTs: OPTION maxrecursion

*Give the numbers from 1 to 999*

```
with numbers(number) as
(select 1 union all
 select number + 1 from numbers
 where number < 999)
select * from numbers
option (maxrecursion 1000);
```

*Maxrecursion is MS SQL Server specific.*



# Application: generate missing months

Xtreme: sales per month in year 2000:

```
select year(orderdate)*100+month(orderdate) mon,  
sum(orderamount) as sales  
from orders o  
where year(orderdate)=2000  
group by year(orderdate)*100+month(orderdate)
```

	mon	sales
1	200002	92130.36
2	200012	167261.28

Problem: not all months occur.

# Application: generate missing months

Solution : generate all months with CTE...

```
with months as
(select 200001 as mon
union all
select mon+1 from months
where mon < 200012)
select * from months;
```

	mon
1	200001
2	200002
3	200003
4	200004
5	200005
6	200006
7	200007
8	200008
9	200009
10	200010
11	200011
12	200012

# Application: generate missing months

Solution: ... and combine with outer join

mon	sales
200001	0.00
200002	92130.36
200003	0.00
200004	0.00
200005	0.00
200006	0.00
200007	0.00
200008	0.00
200009	0.00
200010	0.00
200011	0.00
200012	167261...

```
with months(mon) as
(select 200001
union all
select mon+1 from months
where mon < 200012),

ord(mon,amount) as
(select year(orderdate)*100+month(orderdate), sum(orderamount)
from orders o
where year(orderdate)=2000
group by year(orderdate)*100+month(orderdate))

select m.mon, isnull(amount,0) sales
from months m left join ord o on m.mon=o.mon
```

# Recursively traversing a hierarchical structure

*Db xtreme: give all employees who report directly or indirectly to Andrew Fuller (employeeid=2)*

```
with bosses (boss, emp)
as
(select supervisorid,employeeid from employee where
supervisorid = 2
union all
select e.supervisorid,e.employeeid from employee e
join bosses b
on e.supervisorid = b.emp)
select * from bosses
order by boss,emp;
```

# Recursively traversing a hierarchical structure

*Db xtreme: give all employees who report directly or indirectly to Andrew Fuller (employeeid=2)*

- *the 1<sup>st</sup> step returns all employees that reports directly to Andrew Fuller*
- *Step 2 add the 2<sup>de</sup> “layer” : the report to someone who reports to A. Fuller*
- *Etc.*

# Recursively traversing a hierarchical structure

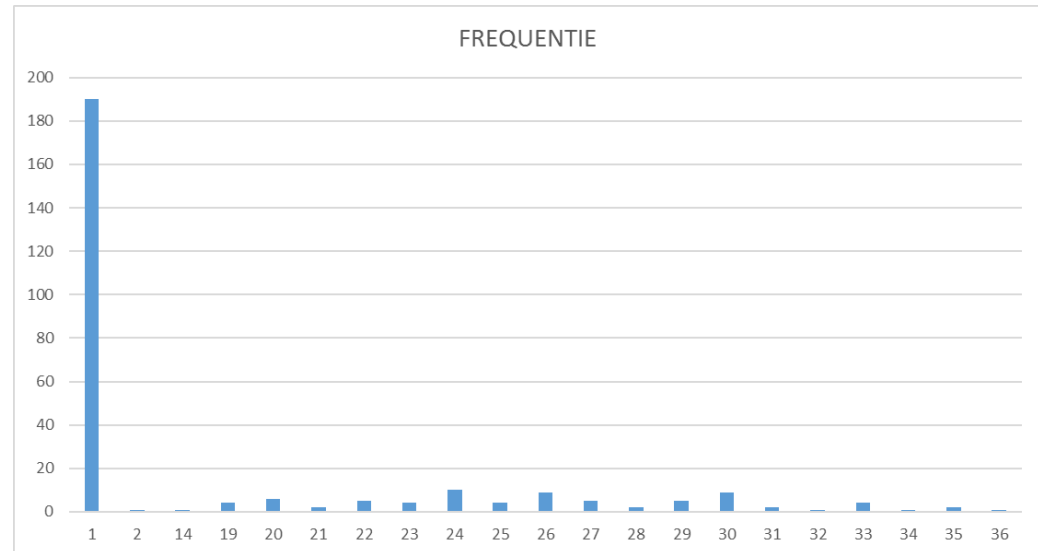
*Db xtreme: give the complete hierarchy of the company, including the names of the employees. Draw the organization chart.*

```
with bosses (symbol,boss, emp,name,path)
as
(select convert(varchar(max),'-----'), supervisorid,
employeeid, lastname+' '+firstname,
convert(varchar(max),employeeid)
from employee where supervisorid is null
union all
select symbol+'-----',e.supervisorid, e.employeeid,
lastname+' '+firstname,
b.path + '.' + convert(varchar(max),e.employeeid)
from employee e join bosses b
on e.supervisorid = b.emp)
select * from bosses
order by path;
```

# Some exercises

## Database Xtreme:

1. Rewrite the “monthly gross margin” example from the subqueries chapter using common table expressions.
2. Make a histogram of the number of orders per customer, so show how many times each number occurs. E.g. in the graph below: 190 customers placed 1 order, 1 customer placed 2 orders, 1 customer placed 14 orders, etc.



# Some exercises

## Execute the script parts.sql in database Xtreme:

3. Show all parts that are directly or indirectly part of O2, so all parts of which O2 is composed.  
Add an extra columns with the path as below:

SUPER	SUB	PAD
O2	O5	O2 <-O5
O2	O6	O2 <-O6
O6	O8	O2 <-O6 <-O8
O8	O11	O2 <-O6 <-O8 <-O11

## Database Tennis:

4. Delete all matches with at least three won sets.
5. Set all won sets to 0 for all players living in Stratford.